

“It ain’t no good if it ain’t
snappy enough.”
(Efficient Computations)

COS 116, Spring 2011
Sanjeev Arora

Administrative stuff

- Readings avail. from course web page
- Feedback form on course web page; fully anonymous.
- Lab 2 due today; HW1 due Thurs.
- Reminder: Lab 3 Wed 7:30 Friend 007. If you are struggling with pseudocode, pls ask TA's in the lab for help.




Discussion Time

What cellular automaton did we encounter in last lecture?

In what ways (according to Brian Hayes) is the universe like a cellular automaton?

What aspect(s) of the physical world are **not** represented well by a cellular automaton?



Today's focus: efficiency in computation

“If it is worth doing, it is worth doing well, and fast.”

Recall: our model of “computation”: pseudocode

Question:

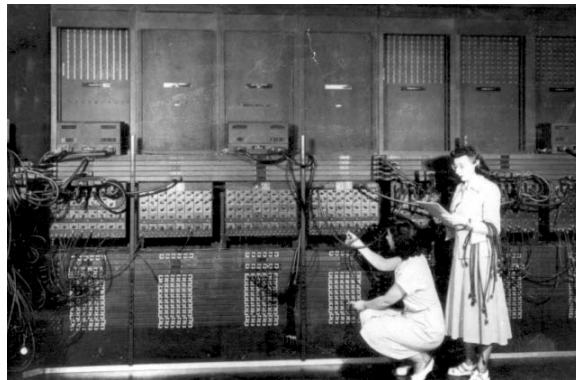
How do we go shopping for a “fast” algorithm?

Like this ?

AMD Phenom™ II quad-core processor 840T;
6GB DDR3 memory; 1TB hard drive;

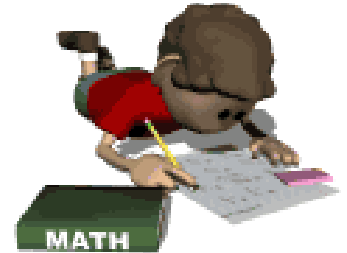
- Ideally, our measure should be independent of:

- machine
- technology



“Running time” of an algorithm

- Definition: the number of “**elementary operations**” performed by the algorithm



- Elementary operations: $+$, $-$, $*$, $/$, assignment, evaluation of conditionals

(discussed also in pseudocode handout)

“Speed” of computer: number of elementary operations it can perform per second (Simplified definition)

- Do *not* consider this in “running time” of algorithm; technology-dependent.

Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- Do for $i = 2$ to n
 - {
 - if ($A[i] < A[best]$) then
 - { $best \leftarrow i$ }}

Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- Do for $i = 2$ to n
 - {
 - if ($A[i] < A[best]$) then
 - { $best \leftarrow i$ }}
- How many operations executed **before** the loop?
 - A: 0 B: 1 C: 2 D: 3

Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- Do for $i = 2$ to n
 - {
 - if ($A[i] < A[best]$) then
 - { $best \leftarrow i$ }
 - }
- How many operations **per iteration** of the loop?
 - A: 0 B: 1 C: 2 D: 3

Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- Do for $i = 2$ to n
 - {
 - if ($A[i] < A[best]$) then
 - { $best \leftarrow i$ }
 - }
- How many **times** does the loop run?
 - A: n B: $n+1$ C: $n-1$ D: $2n$

“iterations”

Example: Find Min

- n items, stored in array A
- Variables are i , $best$
- $best \leftarrow 1$
- Do for $i = 2$ to n

```
{  
    if ( $A[i] < A[best]$ ) then  
    {  $best \leftarrow i$  }  
}
```

1 comparison and maybe an assignment = at most 2 operations per loop iteration

Uses at most $2(n-1) + 1$ operations (roughly = $2n$)

Number of iterations

Initialization

Efficiency of Selection Sort

Do for $i = 1$ to $n - 1$

{

Find cheapest bottle among those numbered i to n

Swap that bottle and the i 'th bottle.

About $2(n - i)$ steps

}

3 steps

- For the i 'th round, takes at most $2(n - i) + 3$
- To figure out running time, need to figure out how to sum $(n - i)$ for $i = 1$ to $n - 1$
...and then double the result.

Gauss's trick : Sum of $(n - i)$ for $i = 1$ to $n - 1$

$$\begin{aligned} S &= 1 + 2 + \dots + (n - 2) + (n - 1) \\ + S &= (n - 1) + (n - 2) + \dots + 2 + 1 \\ \hline 2S &= n + n + \dots + n + n \end{aligned}$$

$n - 1$ times

$$2S = n(n - 1)$$

- So total time for selection sort is $\leq n(n - 1) + 3n$





Discussion Time

“20 Questions”:

I have a number between 1 and a million in mind.
Guess it by asking me yes/no questions,
and keep the number of questions small.

Question 1: “Is the number bigger than half a million?” No

Question 2: “Is the number bigger than a quarter million?” No

Strategy: Each question halves the range of possible answers.

Pseudocode: Guessing number from 1 to n

```
Lower ← 1
Upper ← n
Found ← 0
Do while (Found=0)
{
  Guess ← Round( (Lower + Upper)/2 )
  If (Guess = True Number)
  {
    Found ← 1
    Print(Guess)
  }
  If (Guess < True Number)
  {
    Lower ← Guess
  }
  else
  {
    Upper ← Guess
  }
}
```

Binary
Search

How many times does
the loop run??

Brief detour: Logarithms (CS view)

- $\log_2 n = K$ means $2^{K-1} < n \leq 2^K$
- In words: K is the number of times you need to divide n by 2 in order to get a number ≤ 1

n	16	1024	1048576	8388608
$\log_2 n$	4	10	20	23



John Napier

Running times encountered in this lecture

	n= 8	n= 1024	n= 1048576	n=8388608
$\log_2 n$	3	10	20	23
n	8	1024	1048576	8388608
n^2	64	1048576	1099511627776	70368744177664

Efficiency really makes a difference!

(NB: for large n , $n^2/10$ dwarfs $10n$!)

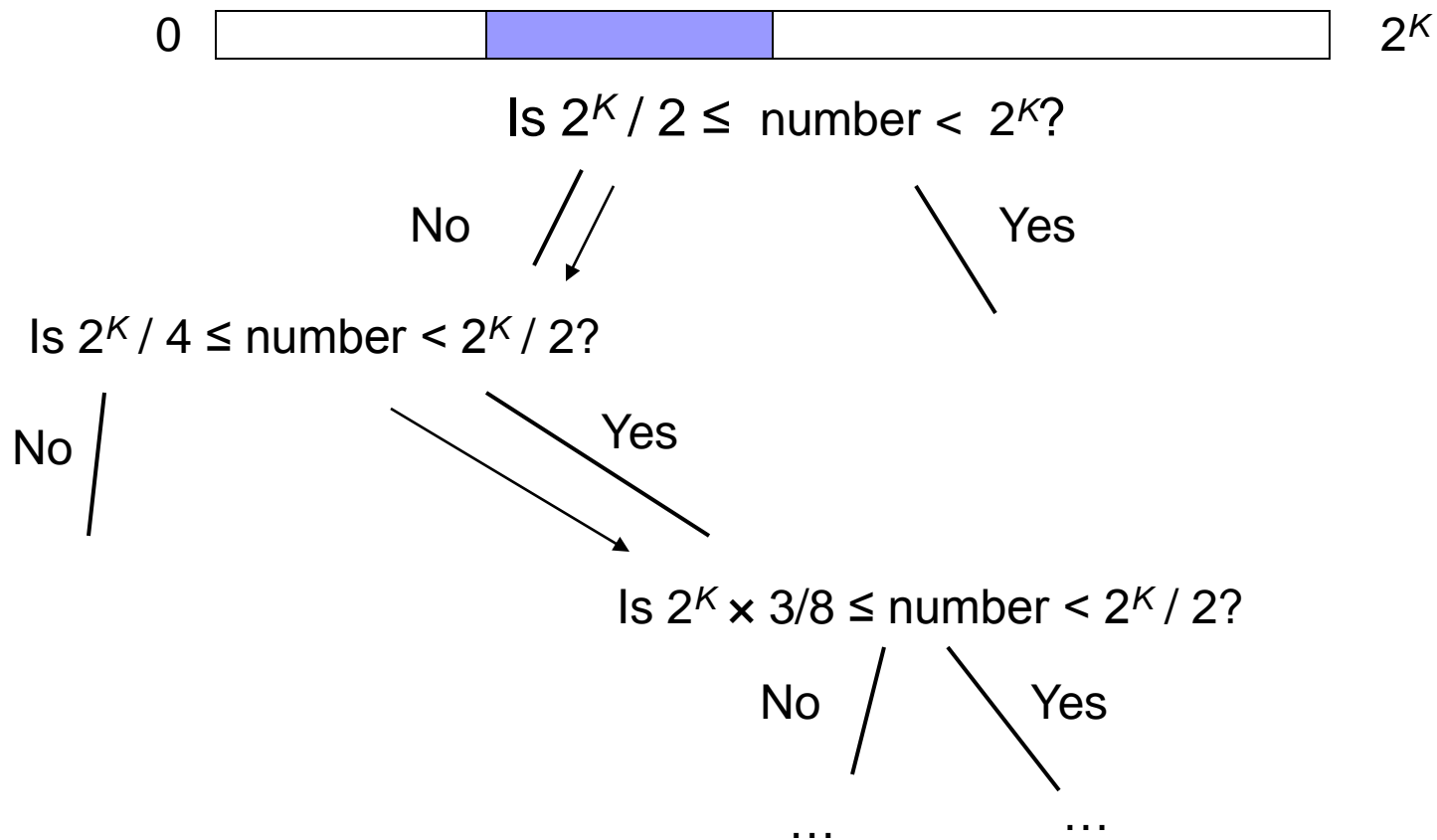


Next....

“There are only 10 types of people in the world – those who know binary and those who don’t.”

Binary search and binary representation of numbers

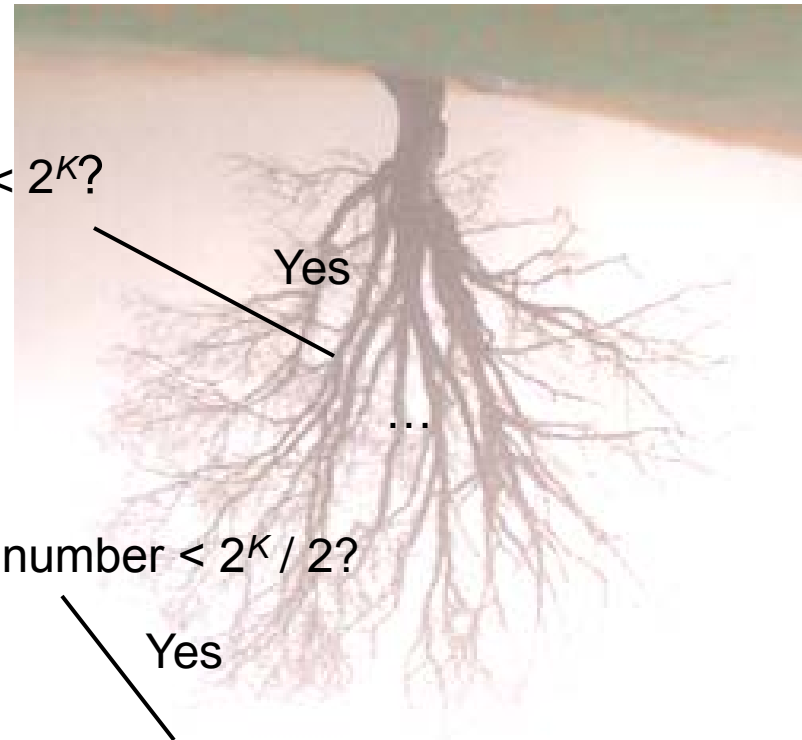
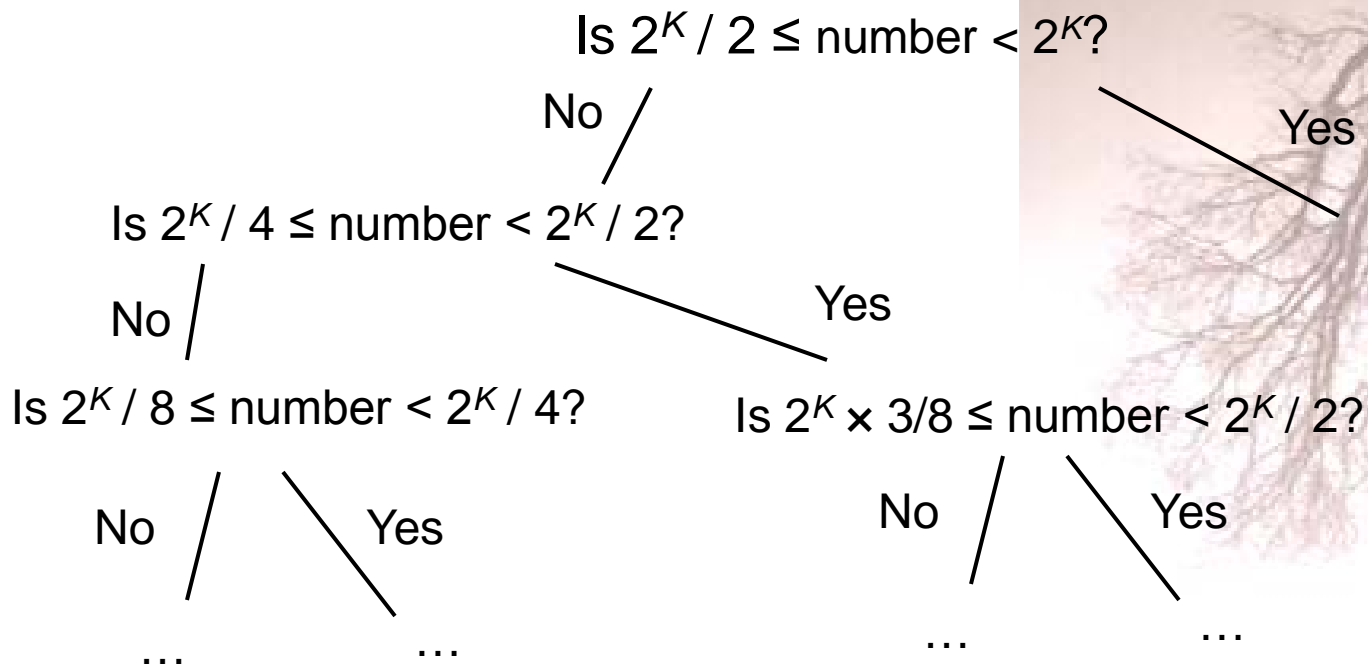
- Say we know $0 \leq \text{number} < 2^K$



Binary representations (cont'd)

- In general, each number from 1 to n can be **uniquely identified** by a sequence of yes/no answers to these questions.
(E.g. $n = 32$; label for 21 = yes, no, yes, no, yes)

- Correspond to **paths** down this “tree”:



Binary representation of n (the more standard definition)

$$n = 2^k b_k + 2^{k-1} b_{k-1} + \dots + 2 b_2 + b_1$$

where the b 's are either 0 or 1)

The **binary representation** of n is:

$$\lfloor n \rfloor_2 = b_k b_{k-1} \dots b_2 b_1$$

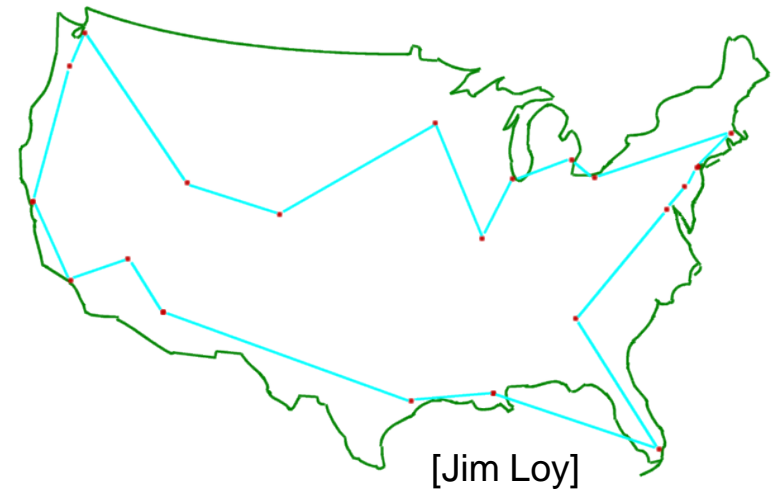
Example: $21 = 16 + 4 + 1 = 2^4 + 2^2 + 2^0$.

Binary representation = 10101

Efficiency of Effort: A lens on the world

- QWERTY keyboard
- “UPS Truck Driver’s Problem” (a.k.a. **Traveling Salesman Problem** or TSP)
- CAPTCHA’s (differentiate humans from machines)
- Quantum computing

(explored in future lectures)



Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

Peter W. Shor[†]

SIAM J.
Computing
26(5) 1997

Abstract

A digital computer is generally believed to be an efficient universal computing device; that is, it is believed able to simulate any physical computing device with an increase in computation time by at most a polynomial factor. This may not be true when quantum mechanics is taken into consideration. This paper considers factoring integers and finding discrete logarithms, two problems which are generally thought to be hard on a classical computer and which have been used as the basis of several proposed cryptosystems. Efficient randomized algorithms are given for these two problems on a hypothetical quantum computer. These algorithms take a number of steps polynomial in the input size, e.g., the number of digits of the integer to be factored.

Can n particles do 2^n “operations” in a single step?
Or is Quantum Mechanics not quite correct?

Computational efficiency has a bearing on physical theories.